Building a virtualized environment for programming courses

Tuisku Polvinen, Timo Ylikännö, Ari Mäkeläinen, Sampsa Rauti, Jari-Matti Mäkelä, and Jani Tammi

University of Turku, Finland
sampsa.rauti@utu.fi

Abstract. Despite computer programming courses today often having hundreds of participants, it is important that the students complete practical programming exercises and have a possibility to participate in hands-on programming sessions, where they can get help from their peers and teaching assistants. However, heterogeneous devices used by students pose a great challenge. When installing and using development tools, both platform specific issues and the limitations of students' personal skill cause issues and lots of extra work for teachers. To address these challenges, we built a virtualized environment that provides the same homogeneous environment for all the students, reducing the time used for configuration and assisting students and allowing time to be used teaching the actual contents of the course. The current study provides an experience report on our course virtualization project, highlighting the drawbacks and benefits of our solution. Our preliminary findings indicate that vast majority of students are using the environment and have found it an effective and usable way to complete the practical coursework.

Keywords: Virtualization, programming courses, CS education

1 Introduction

In computer programming courses with hundreds of participants, using MOOC[1]-like elements such as online reading materials and short lecture videos has become popular. At the same time, it is important from both academic and career perspective that the students complete practical programming exercises and have an option to participate in hands-on programming sessions, where they can get help from their peers and teaching assistants.

On most courses, students are expected to bring their own laptops, on which course exercises are performed, although computer classes are available for those without a laptop. Students' personal devices represent a heterogeneous operating environment for course specific tools and software and as such, it is subject to possible platform specific issues as well as to the limitations of students' personal skills and how well the provided instructions have been understood. This can

[1] Massive Open Online Course

cause extra work for teachers in trying to support the students in getting the required software properly set up or even cause some less experienced students to fall behind without extensive guidance and help from their peers or the teachers. Even when the efforts of the teacher overcome these challenges, valuable time has been lost in issues that do not contribute towards the planned learning outcomes.

To address these challenges, we launched a course virtualization project at the Department of Future Technologies of the University of Turku. The aim of the project is to design a solution that: 1) reduces the time used for software configuration on courses so that the time can be used teaching the actual contents of the course, 2) lessens the burden of teachers of large courses by reducing the amount of assistance students need for using course tools, 3) ensures that every student has access to the same homogeneous environment, 4) and for the course specific software, eliminate the challenges imposed by heterogenous environment of students' devices (e.g. the need to support multiple software versions).

In this paper, we present a case study on building a virtualized environment for several programming courses. The paper provides an experience report describing our virtualization solution, discusses the challenges and pitfalls setting up such a virtual environment may involve, and highlights the achieved benefits.

2 Virtual machines as a solution

To solve the challenges discussed above, pre-configured virtual machines (VM) were built and pre-loaded with course specific software and content and then distributed to students. Each VM runs a Linux operating system (OS) and comes with all needed development tools and programs pre-installed. Installation of a hypervisor is considerably simpler than installing and configuring large sets of development tools, and to further simplify the process, we provided students with clear documentation on how to install and use the VMs. Our approach has several positive effects:

– The teacher has comprehensive knowledge of the environment that students have at their disposal, making it significantly easier to provide technical support. Compared to the case of multiple platforms, the VM reduces most issues to a single problem of starting the hypervisor for the first time.
– The teacher can customize the virtual machine before distributing it to the students. Quick changes to the machines are easily possible if needed.
– Unlike a remote environment, the VM operates on the student's own system, regardless of network connection availability. If the course assignment can be even partially solved offline, this can mitigate potential network issues, especially near crowded WiFi or rooms with connectivity issues [15].
– A student can have several different VMs simultaneously, each serving the needs of a different course. Isolating different course VMs eliminates issues that might result in conflicting versions or configurations. The design also isolates the student's personal system from potentially harmful, but educationally insightful experiments in the development environment.

– Installing a hypervisor and a VM is simple, and advising students in their use takes less time from the teachers than all the installation and configuration help that would be needed without a virtual machine.
– The same process and VM model can be used across several different courses, unifying some of the technical support needs of students. The same unified environment also quickly becomes familiar when used on multiple courses.
– The use of VMs also provides students hands-on experience with virtualization and a low-threshold opportunity to become familiar with Linux and Unix-based operating systems and development environments.

Although cloud and remote desktop technology has been utilized in several other pre-built virtualization solutions in higher education [4, 12], we did not want to implement a primarily cloud-based environment. Our university provides a cloud-based desktop service compatible with our virtual machines and we encourage students who do not possess their own devices to utilize it, but we saw several benefits in operating outside the cloud.

Some of the reasons for distributing detatched VMs include wanting students to be able to fully utilize their own devices, to work on their assignments regardless of internet access and to have full administrator rights in the virtual environment they use. The main advantages of this self-administration is that it provides the students with freedom to modify and experiment with their environments as they like, eliminating the university IT services as a middleman that manages the environments and their privileges.

3 Virtual machine design

3.1 Prerequisites and initial design

We began the VM design by mapping out which courses to include in the requirement analysis. We then proceeded by listing the software, functional, and other requirements of each course. We expected a lot of variation in students' system performance, for example due to legacy hardware, but also modern SSD drives seem relatively small (e.g. 20 GB host OS on a 250 GB disk). For the VMs to work well and to minimize disk footprint, we set an goal to be as small and lightweight as possible. Including all the development tools in a single VM would have doubled its size and added confusion with too many options.

Fig. 1. Our initial VM composition plan.

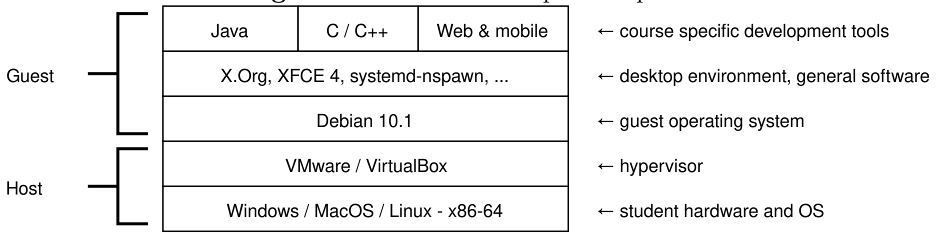
Guest
Host
Java
C / C++
Web & mobile
X.Org, XFCE 4, systemd-nspawn, ...
Debian 10.1
VMware / VirtualBox
Windows / MacOS / Linux - x86-64
← course specific development tools
← desktop environment, general software
← guest operating system
← hypervisor
← student hardware and OS

Table 1. Overview of the major applications included in each different VM setup.

VM	xfce4 desktop	Git	Chromium	Geany	Eclipse	VS Code	OpenJDK	Scene Builder	VisualVM	Dr. Java	Scala	Maven	GCC	Gdb	Clang	Valgrind	Meson	VSC Plugins	Node.js	adb	Firefox	R	Jupyter
Java	×	×	×	×	×		×	×	×	×	×	×											
C/C++	×	×	×	×		×							×	×	×	×	×	×					
Web	×	×	×	×		×													×	×	×	×	×
	General				IDE		Java tools						C/C++ tools						Web			other	

We split the design to a set of VMs with a uniform base and a varying selection of specific development tools (Fig. 1). The requirements for each course were laid out in a table and studied to find similarities between courses to deduce which courses could share a VM setup. The analysis resulted in three different virtual machines: a Java VM, a C/C++ VM, and a VM for web and mobile tools (Table 1).

All of our three initial VMs shared some common software which would have use on all of the courses included in the project, such as a web browser and Git. On the VM intended for use on Java courses, Eclipse IDE[2] was chosen as an IDE because it has been widely used on the respective courses in the recent years. For the C/C++ and Web and mobile setups we chose Microsoft's Visual Studio Code[3] for similar reasons.

The majority of the courses included in the project were Java-related, and while there were some development tools that were not needed on all of them, the requirements were otherwise mostly the same. The benefit of having one Java-oriented VM for use on all Java courses as opposed to two or three slightly different ones was greater than the drawback of the tools taking slightly more disk space on the VM, so we decided to include them all.

In the Web and mobile setup, we included an additional browser for web development purposes, and Android Debug Bridge (adb) to enable the user to connect to an Android device for mobile development. We also included Python and R tools to broaden the usability of the setup to cover some additional courses.

In the C/C++ setup, in addition to the IDE, installed software mainly included libraries, compilers and debug tools. Since the C/C++ and Web and mobile setups both share the same IDE, and including the software from both setups in one VM would not result in an unreasonably large image, we have considered combining the two in the future to further reduce the amount of different VM setups.

Selecting a hypervisor In the initial development and testing phase of this project, two popular type-2 hypervisors for desktop use, VMware Workstation

[2] https://www.eclipse.org/

[3] https://code.visualstudio.com/

Table 2. Comparison between VMWare and VirtualBox.

	Pros	**Cons**
VMware	simple & easy user interface stable, backward compatible	Requires commercial license on MacOS, closed source
VirtualBox	free and open source multi-platform	more complex setup stability issues

[19] and Oracle VirtualBox [14], were evaluated. Both virtualization solutions provided all the needed functionality (fast full screen graphics, sound, network, USB, and web camera support). The practical aspects of using both are compared in Table 2. Creating and managing the virtual machines with VMware software was considerably simpler, as the kernel drivers are part of mainline Linux and the userspace guest tools are included in Debian's repositories. The VMware Player on Windows and Linux is free for personal use, while the VMware Fusion version for MacOS requires a commercial license.

VirtualBox is free and open source, eliminating any licensing issues. Starting with VirtualBox 6, the hypervisor switched to the mainline VMware graphics driver, but the other drivers and userspace tools required a separate installer, which complicates the VM generation more than its use. Moreover, a non-free extension installer is also available with further drivers. Earlier comparisons conclude that the differences between VirtualBox and VMware are not significant performance-wise [5]. At first we opted for the open source option, but found out that on MacOS machines there was a consistent problem with VirtualBox crashing every time when closing the application. The MacOS version also had problems adjusting the resolution correctly and did not support the sharing of the clipboard. Setting up the guest utilities required a bit patching in the build process. Earlier research indicated that others had similar issues, at least with the previous VirtualBox versions. Many who had initially chosen VirtualBox later switched to other solutions, most notably VMware [10, 16].

For insights in how users experience these hypervisors and their use, we conducted a small testing session with a group of 7 students. Using both hypervisors, the students in our test group were tasked to install the hypervisor and perform the same set of tasks on them. The tasks included importing an existing VM, powering the VM on, switching to full screen mode and back as well as turning off the VM while being recorded. The students were also asked to give feedback after the session. While the differences were not significant, a few students had a preference for VMware Player's simpler GUI.

Selecting the guest OS Linux was chosen as the VM guest OS to avoid the difficulties with redistribution, license costs, and vendor lock-in as we can freely choose to include any software with permissive or copyleft licenses and make customizations adhering to these principles. The licensing is especially problematic for courses being offered to external organizations and non-students.

Table 3. The initially considered Linux guest OS distributions.

Distribution	**Pros**	**Cons**
Alpine	lightweight, customizable, secure	libc incompatibilities
Arch	customizable, variety of packages	rolling-releases & maintenance
CentOS	stable package selection	up to 5 year (major) release cycle
Debian	stable, long-term support	2 year (major) release cycle
Fedora	developer-friendly	large disk and memory footprint
Gentoo	extremely customizable	maintenance, expertise required
OpenSUSE	stable packages, minor versions	2–3 year (major) release cycle
NixOS	declarative reproducible builds	support, expertise required
(X)Ubuntu	beginner-friendly, documentation	large disk footprint

The first challenge with Linux is to find the most suitable distribution for the task at hand. The VM could be seen as a new distribution, but we wanted to leverage existing work as much as possible, while offering a streamlined and polished user experience and low resource consumption. We experimented with the default installations of 10 common distributions (Table 3) and compared their customizability, disk footprint, release model, and software compatibility.

At first, we settled for Ubuntu 18.04 LTS, valuing its beginner-friendliness, documentation, and support[1]. Ubuntu had a large default installation and many unnecessary packages needed to be removed by hand to attain our primary goal of small size. Debian, on which Ubuntu is based, was initially discarded due to its outdated support of Java and Python. Debian 10 was released during the development, provided a simple migration from Ubuntu, equally fulfilled the requirements, while using far less space, which led us switch the OS.

3.2 Build process

Automation of installation While creating several similar VMs or VM versions, manual labor increases the risk of mistakes and takes a lot of time [17]. Our aim was to produce reproducible builds with automated installation tools.

We considered multiple ways of initiating the OS installation: 1) default OS installer 2) preseed[4] with a HTTP server and one of the hypervisors 3) preseed with virt-install[5] & QEMU[6], and 4) debootstrap[7]. The preseed approach offered basic configuration of the OS settings, but performed slower and required either a Linux host or a server setup to inject the custom configuration. Debootstrap can be containerized, virtualized, performs the best with a tiny set of core packages, but fails to install the whole OS. It also leaves the system largely unconfigured.

For building the OS core, we picked debootstrap. The phase sets up the core utilities and the package manager. While other installation options seem more complete, the advantage of debootstrap is very little distribution specific configuration. Instead, we use Ansible for configuring the rest of the system.

[4] https://www.debian.org/releases/stable/armhf/apbs02.en.html
[5] https://packages.debian.org/buster/virtinst
[6] https://www.qemu.org/
[7] https://packages.debian.org/buster/debootstrap

Ansible playbooks The next stages of installation were carried out with Ansible playbooks, first to produce the platform image and desktop environments (as depicted in Fig 1), then the course specific applications. The playbooks first install all the platform packages, then set up standard system-wide (timezone, login manager, users, etc.) and user-specific configuration (locale, key layout, etc.), usually carried out during the OS installation. All the VM specific polish and customization is performed along the generic standard configuration.

A second set of playbooks installs and sets up the course and application dependent applications. The Ansible tasks were divided among smaller modular playbooks which were imported in the three management playbooks for each VM version (Java, C, web & mobile programming). This way, it can be ensured that all of the common tools were installed the same way on each different machine, while streamlining the editing process so that an edit in one playbook takes effect in every management playbook in which it is imported.

Finally, further playbooks were used for removing unnecessary suggested packages and disabling services when the requested packages have been set up. Since we start the distribution creation from scratch, most of the time we want the suggested packages, but in few hand-picked cases they are unnecessary.

The approach we picked allows a highly customizable setup with easy configurability by defining a selection of playbooks that define the distribution. For instance, one playbook might define a Java runtime environment, and the other the Java tools based on a specific development environment.

Since the final Ansible playbooks phases clean up the filesystem and remove all intermediate states of the build process, the end result is a production ready, bootable virtual machine that can be exported as a standard OVF/OVA (Open Virtualization Format/Appliance) archive. The archive is compressed which further decreases the size of the distribution as the user storage and swap space are initially empty. Since we only use free and open source components in the virtual machine, we can freely distribute the machine. Our scripts and instructions for building the VM are also released as open source via the university home page.

3.3 Distribution customizations

Desktop modifications To further accommodate the virtual machines, we decided to make changes to the default desktop layout. We decreased the number of workspaces in Xfce to one (similar to classic Microsoft Windows) and included the most significant development tools as desktop shortcuts, even if they could also be found from the application menu. Most students using our virtual machines are Windows users, and to make the environment feel more familiar to them, we removed the second default Xfce panel and moved the taskbar from its default top position to the bottom of the screen.

Startup scripts In addition to the desktop modifications, the most apparent modification we developed is the script that runs when a student logs in to their virtual machine for the first time. The script configures and performs the followings tasks: 1) localization (language, timezone, etc.) 2) keyboard layout 3) generation of the SSH key to be used with the university's GitLab 4) configures the local Git client to make use of the user's name and email.

Help and support To provide students using our VM with helpful resources and documentation, we prepared bookmark and PDF files that included useful links and guides. The bookmark file was designed to be inserted automatically to the VM's default browser, and the PDFs to the Documents folder.

4 Testing and results

4.1 Performance tests

In addition to the UX testing, we ensured with preliminary technical tests and benchmarks that the VMs would perform in the intended purpose

Memory and disk footprint In a low-memory test, the guest OS was limited to 0,5 GB of RAM and 2 GB of swap space. The VM started to slow down considerably, but did not crash. The total memory use reached 1,5 GB when compiling a non-trivial Java course assignment. We also installed VMs on different hosts and experimented with a variety of course assignments and the recommended amount of 2 GB of RAM. No particular problems arose during the test. The final September 2019 release of the Java, C/C++, and web/mobile VMs required 2685, 2655, and 2829 kB of disk space for the OS. In addition, a 2 GB partition was reserved for swap. The initial user home partition was left empty.

Performance Geekbench[8] is a widely used synthetic CPU performance benchmark software that performs various single and multi-core workloads from different domains. We compared the host and guest GeekBench 5 results on all the supported operating systems (Table 4). As expected, the single-core VM performance was generally 3–9% lower for both hypervisors. To our surpise, on Windows VirtualBox performed 28% (SC) and 24% (MC) worse than VMware. On MacOS, VirtualBox performed only around 1% faster, which led us to believe it did not use multiple host cores even when configured in such a way.

[8] https://www.geekbench.com/index.html

Table 4. GeekBench 5 single (SC) and multi-core (MC) results on different hosts and hypervisors. Hypervisors were allocated two cores (our production VM configuration).

	Host		**VMware**		**VirtualBox**		**CPU (cores + HT)**
Linux	759	4010(12)	690	1353(2)	692	1329(2)	Intel i7-3970X (6+6)
Windows	1071	3172(8)	1039	1813(2)	751	1372(2)	Intel i7-8550U (4+4)
MacOS	1005	5020(6)	1093	2138(2)	963	977(2)	Intel i5-8500 (6)

4.2 Qualitative tests and user experience

We also tested the proper functioning of virtualized hardware such as network, sound input/output, keyboard and mouse integration, system services, startup scripts, and shared clipboard and folders. No major issues were found while carrying out these tests. VirtualBox on MacOS was unable to get shared clipboard and file drag'n'drop working. Few students who chose VirtualBox on Mac instead of the recommended VMware also failed to adjust the guest screen resolution.

We organized tutorial sessions for setting up the VM along normal course exercise sessions. Based on the questions presented in both the sessions, getting used to the basic functionality of the selected guest OS did not pose problems. On the host side, some devices required manually enabling the virtualization in the BIOS settings. We originally expected issues with the requirements for 64-bit CPUs, the VT-x processor capability flag, and the minimum amount of 2 GB of RAM, but did not encounter any systems with lower specifications.

5 Discussion

Projects similar to ours have been realized in other universities. Desktop virtualization is a common theme in the literature on education, and it is being used for e.g. computer security and networking [9, 8, 18], operating system courses [13], and some focus in computer science education in particular [6, 10, 16]. Some users also provide virtual research tools for a specific field [7, 2, 11, 3].

While our virtualized environment was only taken into use recently (in September 2019), the initial results were promising. In our tutorial sessions, the majority of students did not need assistance in using the VM. We believe the reasons for these are the practical help given to students in the sessions, simplicity of the VM, the customizations we made to simplify the user interface, and the Student's Guide we have composed. We observed that the problems caused by diverse platforms, devices, and tool installation problems also significantly decreased.

In exercise sessions, we observed that ∼75% of the attendees used the VM for the assignments. We offered a public VM download link, and collected the number of downloads without tracking unique users. The number of downloads was 97% of the size of the active student pool (n=136) and rose to 177% by the end of the course. We think the reason for this were the two bug fix releases we did during the course as the first version had minor tool configuration problems. Rest of the students, some who are well versed technically, probably had the tools already installed or managed to install them natively on their systems.

The downside of the pre-installed VM approach is that students do not need to install and configure the tools anymore unless they want to. We wanted to leave it for each teacher to evaluate if these tasks are relevant for the learning outcomes, or if the they are merely an undesirable time expenditure. Our solution can offer advantages also for gaining practical experience in installation and configuration as well, as it allows students to experiment freely and rollback actions - either by using snapshots feature or simply reimporting the whole instance and starting from a clean slate again. This can negate issues usually associated with diverse platforms and devices.

Overall, we experienced less problems inthe VM deployment and use than reported earlier [16, 10]. Therefore, we can draw a cautious conclusion that the technology and hypervisors appear to have matured to be effectively employed to help in IT education. In our use case, the virtualization was used for programming exercise projects and the majority of our students major in IT, which might affect the results positively and may not generalize to other subjects (even in IT) such as system administration or cyber security. However, at least for our programming courses, the developed environment can be considered a success.

6 Conclusion and future work

In this paper, we have proposed a virtualized environment to make practical parts of programming courses easier. We have described the process of building the environment and its benefits and challenges. The system is currently being used on programming courses with hundreds of participants. Our preliminary findings indicate that vast majority of students are using the environment and have found it an effective and usable way to complete the practical coursework. In future, we aim to further study students' user experiences when using the system, as well as perceived benefits and drawbacks of the virtualized environment. We also plan to write an extended article that covers the technical aspects of our virtualized environment in more detail.

References

1. Al Housani, B., Mutrib, B., Jaradi, H.: The Linux review - Ubuntu desktop edition - version 8.10. In: 2009 Int. Conf. on the Current Trends in Inf. Tech., IEEE (2009)
2. Angiuoli, S.V., Matalka, M., Gussman, A., Galens, K., Vangala, M., Riley, D.R., Arze, C., White, J.R., White, O., Fricke, W.F.: CloVR: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing. BMC bioinformatics **12**(1) (2011)
3. Dahlö, M., Haziza, F., Kallio, A., Korpelainen, E., Bongcam-Rudloff, E., Spjuth, O.: Biolmg.org: A Catalog of Virtual Machine Images for the Life Sciences. Bioinformatics and Biology insights **9** (2015) 125–128
4. González-Martínez, J.A., Bote-Lorenzo, M.L., Gómez-Sánchez, E., Cano-Parra, R.: Cloud computing and education: A state-of-the-art survey. Computers and Education **80** (2015) 132–151 Cited By :131.
5. Horalek, J., Svoboda, T.: Analysis of Virtualization Tools for Education Purposes. Journal of Telecommunication, Electronic and Computer Eng. **10**(1-8) (2018)
6. Ketel, M.: A virtualized environment for teaching IT/CS laboratories. In: Proceedings of the 48th Annual Southeast Regional Conference, ACM (2010)
7. Kind, T., Leamy, T., Leary, J., Fiehn, O.: Software platform virtualization in chemistry research and university teaching. Journal of Cheminformatics **1**(1) (2009)
8. Li, P.: Integrating virtualization technology into remote lab: A three year experience. In: American Society for Engineering Education (ASEE). (2009)

9. Lunsford, D.L.: Virtualization technologies in information systems education. Journal of Information Systems Education **20**(3) (2009) 339–348
10. Malan, D.J.: From cluster to cloud to appliance. In: Proceedings of the 18th ACM Conf. on Innovation and technology in computer science education, ACM (2013)
11. Metze, F., Fosler-Lussier, E., Bates, R.: The Speech Recognition Virtual Kitchen. In: 13th Annual Conf. of the Int. Speech Communication Association. (2013)
12. Moser, S., Krapp, F., Bärtele, S., Wunderlich, K., Gröger, G., Slomka, F., Schumacher, H.: Cloud-based virtual desktop environment for advanced online master's courses. (2015) cited By 3.
13. Nieh, J., Vaill, C.: Experiences teaching operating systems using virtual platforms and linux. ACM SIGOPS Operating Systems Review **40**(2) (2006) 100–104
14. Oracle: Oracle VM VirtualBox. https://www.virtualbox.org/
15. Paulus, S., Smits, T., Becht, T., Kol, S.: Ubiquitous learning applied to coding: A set of tools and services to deliver code-intensive learning contexts to student devices. In: Proceedings of the 3rd European Conference of Software Engineering Education. ECSEE'18, New York, NY, USA, Association for Computing Machinery (2018) 87–92
16. Sayler, A., Grunwald, D., Black, J., White, E., Monaco, M.: Supporting CS education via virtualization and packages. In: Proceedings of the 45th ACM technical symposium on Computer science education. SIGCSE '14, ACM (03 2014) 313–318
17. Sun, C., He, L., Wang, Q., Willenborg, R.: Simplifying service deployment with virtual appliances. In: 2008 IEEE Int. Conf. on Services Computing, IEEE (2008)
18. Syamsuddin, I.: A virtual lab model to integrate computer networking courses. In: 2nd Int. Conf. on Education, Science, and Technology, Atlantis Press (2017)
19. VMware: VMware – Official Site. https://www.vmware.com/